METHOD AND COMPUTER SYSTEM FOR DOCUMENT AUTHORING

Field of the Invention

The present invention generally relates to electronic
5   data processing, and more particularly, relates to
methods, computer program products and systems for
document authoring.

Background of the Invention

10   Some software development platforms, such as the
Eclipse Platform, are designed for building integrated
development environments (IDEs) that can be used to
create applications as diverse as web sites, embedded
JavaTM programs, C++ programs, and Enterprise
15   JavaBeansTM. Although the Eclipse Platform typically
have built-in functionality, most of that functionality
is very generic. It takes additional tools to extend
the Platform for handling new content types, new
functionality for existing content types, and to focus
20   the generic functionality on specific tasks.
        The Eclipse Platform is built on a mechanism for
discovering, integrating, and running modules called
plug-ins. For example, a tool provider can write a tool
as a separate plug-in that operates on files in the
25   workspace and surfaces its tool-specific user interface
(UI) in the workbench. When the Platform is launched, a
developer (also referred to as author) is presented
with an IDE composed of the set of available plug-ins.
        More and more heterogeneous devices access
30   application servers running applications developed by
using an IDE. Current IDEs support the development of
user interfaces for applications that were originally
foreseen to interact with homogenous delivery context
(e.g., a screen of low resolution, such as 800x600
35   pixels). Developers have to adapt application user

interfaces for different types of delivery context.
This task becomes increasingly difficult with the prior
art IDEs having a lack of specific support for device
independent development of user interface documents.

5   Some web-development tools, such as DreamWeaver
from Macromedia Inc., provide tools for XML validation,
browser pre-visualisation and code completion. However,
there is a lack of support for transforming device
independent representations into various target

10   languages, such as WML or VoiceXML. Further, where
device independent development is enabled it lacks
support for visualization of the results.

Summary of the Invention

15   The present invention provides computer system, method
and an integrated development environment according to
the independent claims for improving the support for
device independent authoring of user interface
documents.

20   The various embodiments of the invention provide
alternative solutions to support a document author to
improve and accelerate the generation of a device class
independent user interface document by generating
device class specific information and providing it to

25   the author.
     In one embodiment according to claim 1 the author
can get information about the complexity of the user
interface by device class through a complexity
indicator.

30   In an alternative embodiment according to claim 10
the author can see how many pages are generated on the
basis or the document for which device class in a
device class dependent view.
     In a further alternative embodiment according to

35   claim 11 the author can get information about the

layout for the various device classes in a frames
layouting view.

Each of the alternative embodiments supports the
author to identify problems in a user interface
5   document related to device class specific restrictions
already during the development of the document.
Identifying such problems at early stages of the
development usually minimizes efforts for adjusting the
device independent document to better comply with the
10 various device class specific restrictions. A result of
the device specific document analysis with the above
embodiments can also be that a user interface document
cannot be used at all by devices belonging to a
specific device class. In this case the author may not
15 release the document for the specific device class.

The aspects of the invention will be realized and
attained by means of the elements and combinations
particularly pointed out in the appended claims. Also,
the described combination of the features of the
20 invention is not to be understood as a limitation, and
all the features can be combined in other
constellations without departing from the spirit of the
invention. It is to be understood that both the
foregoing general description and the following
25 detailed description are exemplary and explanatory only
and are not restrictive of the invention as described.

Brief Description of the Drawings

FIG. 1 is a simplified block diagram of an integrated
30   development environment (IDE) for generating
         user interface documents according to the
         invention;

FIG. 2 illustrates an implementation of a selection
         screen of a template wizard that can be used
35   with the IDE;

FIG. 3 is one implementation of the main window of the
       IDE including an explorer/navigator view and an
       editor;

FIG. 4 shows an implementation of the IDE main window
5   further including a tree-based outline editor
       and a fragment repository;

FIG. 5 illustrates the IDE main window when further
       including a device dependent frames layouting
       view;

10 FIG. 6 shows the IDE main window further including a
       device class dependent page view;

FIG. 7 shows the editor when combined with a Java
       filtering tool in the IDE main window;

FIG. 8 illustrates details of a device class dependent
15   complexity indicator as being part of the IDE;
       and

FIG. 9 shows an implementation of a complexity display
       when integrated into the IDE main window.

20 Detailed Description of the Invention

FIG. 1 is a simplified block diagram of an integrated
development environment 999 (IDE) that can be used for
the development of user interface documents. The IDE
can be implemented as a computer program running on a
25 computer system that includes one or more computing
devices. The IDE 999 includes a general tool set 100
and a device class dependent tool set 120 that is
integrated 990 with the general tool set 100. Examples
of tools included in the general tool set are an editor
30 104 and an adaptation engine 105. Examples of tools
that can be included in the device class dependent tool
set 120 are a device class dependent page view 122, a
fragment repository 123 supporting device dependent
fragments, a device dependent frames layouting view 124

and/or a device class dependent complexity indicator
121. The tools are explained in detail in the later
description.

5   The following description describes by way of
example the development of a user interface document on
the basis of a Renderer Independent Markup Language
(RIML) document by using the IDE 999. However, the
present invention can be applied to any other document
type, such as documents written in Hypertext Markup
10   Language (HTML), Extensible Markup Language (XML),
Java, etc. RIML is an XML based markup language. The
user interface document can be stored in form of a file
or any other suitable data structure.

The IDE 999 can include a variety of further tools
15   that support the author of the document to reduce the
development time for document development and to detect
and correct errors within the document. The device
class dependent tools 120 provide special support for
the development of documents that are used in mobile
20   applications and, therefore, need to be compatible with
a variety of device classes. A device class includes a
plurality of restrictions that are typical for devices
(e.g., mobile devices) belonging to the device class.

The first step in document development is usually
25   to create a development space within the IDE 999, where
the document(s) can be assigned to. For example, when
using the Eclipse platform as IDE, an Eclipse project
can be created for working on RIML files.

A device independence plugin installation folder
30   can include several folders. Each of the folders stores
information for a specific tool of the IDE.

For example, a template folder can include RIML
templates available within the authoring environment to
be used by a template wizard. A quickhelp folder can
35   include XML description files used by a quickhelp tool

for providing help information related to a tag on
which a cursor is positioned. A fragments folder can
include RIML fragments available within the authoring
environment, plus descriptions of the respective

5   fragments. A schemas folder can include XSD grammar
required by a RIML validation and code completion tool
to work.

The various tools of the IDE and how they interact
to improve device independent authoring are described

10   in the following.

FIG. 2 illustrates an implementation of a selection
screen of the template wizard 106 to support the
document author to avoid starting from scratch when

15   creating 502 a new document 300. Predefined document
templates RIML1 to RIML3 allow the author to reuse
usability approved templates and modify them. For
example, the templates can be RIML documents which
contain a specific layout, and which may be tailored

20   for a specific application domain, user group or
various device classes.

For example, after having selected RIML
(illustrated by an ellipse) in a New-section of the
selection screen to indicate that the author intends to

25   create a RIML document, the author can select 501 a
template RIML3 in a From-Template section of the
selection screen. The selected template RIML3 is then
loaded into the editor 104, where it can be saved as
the new document 300.

30   The selection function as described can be
integrated in the IDE UI or displayed on a popup window
prompting the author. By checking target user group(s),
application domain and device classes for which the
application is to be tailored, the selection screen can

35   be used to filter the available templates to display

only templates that comply with the selection criteria
entered on the selection screen. In case no template
matches the selection criteria, a default template can
be selected. The default template includes at least
5   correct RIML element attributes and basic mandatory
elements such as head and body. The template can then
be stored in the corresponding development space of the
IDE (e.g., Eclipse project) under a file name defined
by the author. In the following it is assumed that the
10   development space is a RIML folder.

FIG. 3 is a possible implementation of the main window
of the IDE 999. The IDE can further include an
explorer/navigator view 107 for visualizing the files
15   that are assigned to the RIML folder. The
explorer/navigator view 107 can be implemented similar
to the Package Explorer in the Eclipse platform.
        The explorer/navigator view 107 is interfaced to
the editor 104 so that the document file 300 can be
20   directly loaded 503 into the editor 104 through a
corresponding interaction of the author with the
explorer/navigator view. For example, the author may
double click on the file name or select a corresponding
open function from a menu or toolbar of the
25   explorer/navigator view. The editor 104 supports
typical functions, such as, text insertion, copy,
paste, cut or syntax colouring. For example, the editor
can 104 be implemented on the IDE main window or in
separate window.
30   The author can select a perspective, e.g., by
selecting a corresponding function from a menu or
toolbar of the explorer/navigator view 107. A
perspective is a set of tools associated with an
extension (e.g., the riml extension of RIML document
35   files), and it defines a certain layout for these

tools. For example, In the Eclipse IDE, one perspective
is associated with one plugin, wherein a plugin can be
composed of several views.

5   The editor 104 can be further interfaced to a
template XML description file including information
about the different available document templates, such
as for example, meta data about device classes
supported by the templates.

10   Source code example 1 shows a template XML
description file that includes a root element named
RIMLTemplate, and which has RIMLTemplate elements, such
as DeviceClass, ApplicationDomain, TargetdUserGroups
and Keywords as children. Source code example 1 shows
the content of the RIMLTemplate node for a template
15   called V2test_pag_pagcol.riml.

```
Source code example 1:
<RIMLTemplate>
     <Path>./V2test_pag_pagcol.riml</Path>
     <Name>V2test_pag_pagcol.riml</Name>
     <DeviceClass>1</DeviceClass>
     <DeviceClass>2</DeviceClass>
     <DeviceClass>3</DeviceClass>
     <DeviceClass>4</DeviceClass>
     <DeviceClass>7</DeviceClass>
     <ApplicationDomain>Enterprise</ApplicationDomain>
     <TargetedUserGroups>Expert</TargetedUserGroups>
     <TargetedUserGroups>Professional</TargetedUserGroups>
     <Keywords>sample paginated column</Keywords>
</RIMLTemplate>
```

The source code 1 XML description file includes
20   information about:
- The path and filename relatively to the IDE root
folder,

- the name of the template (here name can mean label, because the actual file name is included in the path element),

- the different device classes handled,

5   - the application domain concerned, and

- the targeted User Groups targeted by this template.

The IDE can further include a link to a usability guideline tool that allows the author to access
10   usability guidelines. For example, usability guidelines, such as the Consensus usability guidelines, are available through the Internet or locally. The usability guideline tool can be interfaced to the editor 104 to be used as an authoring help. For
15   example, a corresponding hyperlink 108 in the IDE can guide the author to the usability guideline tool.

The IDE can further include a context sensitive help tool to provide to the author a quick info about a specific tag used in the document 300 when loaded into
20   the editor 104. The information can include a short description of the tag's function and corresponding hyperlinks to the RIML usability guideline. Hyperlinks proposed by the context sensitive help tool are the ones considered as relevant with respect to the edited
25   RIML document from a usability perspective. For example, the context sensitive help tool can be called through a specific function key or a corresponding IDE menu entry or other control element. When evoking the context sensitive help tool, for example, a popup
30   window can show the tag's description and the links to the usability guideline.

The IDE can further include a quickhelp XML description file. The quickhelp file can include the
35   relevant content for the help on tags. Source code

example 2 shows a quick help file that includes a root
element called tags having child elements. In the
example, the name of a child element names is the name
of the tag concerned.

5

source code example 2:

```
<?xml version="1.0" encoding="UTF-8"?>
<tags xmlns="http://not-real"
     xmlns:riml="no-need-to-edit-this1"
     xmlns:smil="no-need-to-edit-this2"
     xmlns:html="no-need-to-edit-this3"
     xmlns:eccdc="no-need-to-edit-this4"
>


<riml:layout>
     <description>
```

All RIML layout definition is enclosed by the layout
element. The layout element has one child, which can be
either be a frame element or a RIML layout container
element.

```
     </description>
     <link>
         <label>Display - Text</label>
         <URL>http://apg.
vienna.org/APGDetail.php?id=126</URL>
     </link>
     <link>
         <label>Page Types - Relationships</label>
         <URL>http://apg.
vienna.org/APGDetail.php?id=194</URL>
     </link>
     <link>
         <label>Spacing in dialogs</label>
         <URL>http://apg.
vienna.org/APGDetail.php?id=308</URL>
```

```
        </link>
        <link>
            <label>Layout conventions</label>
            <URL>http://apg.
vienna.org/APGDetail.php?id=311</URL>
        </link>
        <link>
            <label>Relationship of elements</label>
            <URL>http://apg.
vienna.org/APGDetail.php?id=315</URL>
        </link>
    </riml:layout>
```


        The IDE can further include a code completion tool
    102. The code completion tool 102 proposes different
    possibilities for auto-insertion of text (e.g.,
5   </riml:layout>, <riml:column>, <riml:row>, <riml:grid>)
    in the editor 104 dependent on the context at a
    specific position (e.g., <riml:layout) within the
    document 300. For example, the specific position can be
    defined in the editor 104 through the cursor position.
10 Such a code completion tool 102 is described in the
    European patent application 02000106.1 published under
    the publication number EP1326175. For example, the
    author can invoke 504 the code completion tool 102 from
    within the editor 104 by using a specific control key
15 combination, such as CTRL+SPACEBAR. A popup window 102
    can then display the different possibilities for
    completion. These possibilities can rely on schemas
    defined in the html root element of the document 300,
    to propose only valid completion options. The author
20 selects one of the proposed completion texts (e.g., by
    double-clicking) to trigger the insertion of the text.
    The code completion tool 102 can also be applied to
    attributes within an element.

FIG. 4 shows an implementation of the IDE main window
when the IDE 999 further includes a tree-based outline
editor 109 for generating an outline view 209 of the

5   edited document 300, such as an XML tree view 209 of
the RIML document 300. The tree-based outline editor
109 further can make editing proposals for the document
using popup windows similar to the code completion
tool. The tree-based outline editor 109 is interfaced

10 with the editor 104, in the sense that when selecting
an element 209' in the outline view 209, the editor 104
highlights 504 the corresponding text portion 309 of
the document 300.

The author can see a list of elements that can be

15 added as children to a current element (e.g., the head
element) and further a list of attributes that can be
added to the current element by selecting the current
element of the outline view 209. A popup window can
propose similar completion as the code-completion tool.

20 When the author selects a completion it will be added
to the tree of the outline view 209 and other views as,
for example, the editor 104 view, will be updated
accordingly.

The IDE 999 can further include a fragment

25 repository 123 for supporting the reuse of fragments of
RIML documents. The fragment repository 123 allows the
author to load and save identified RIML fragments. When
the IDE 999 is started, the fragment repository 123 can
be visualized as a tree structure 223. The fragment

30 repository includes fragments already saved in the
past. The author creates the structure of the fragment
repository when deciding where to save the different
fragments. For example, the author may choose the
hierarchy of the RIML document, which may be decomposed

35 into the different tasks. It may be decomposed into

different versions of a search result based on the
various device classes DC1, DC2.

　　　The author can save the whole head part of a RIML
document as a root node that contains all layout
5　　information. Then, layout parts that are specific to
corresponding device classes DC1, DC2 can be saved as
child fragments. Thus, if a further author wants to
have a layout that is specific to a single device
class, he/she can directly use the child fragment that
10　is specific to the desired device class DC1, DC2 layout
instead of using the root node containing also
information that is not necessary for the desired
device class.

　　　The fragment repository has at least the functions
15　"save fragment" and "load fragment".

　　　To save a fragment, the author highlights the
desired fragment in the editor (e.g., through a mouse
or a keyboard). The fragment can be any part of the
document text (e.g., text portion 309). For example it
20　can be but does not have to be a valid XML fragment.
Then, the author selects the node the fragment
repository view where the fragment is to be appended
to. The author can enter a name and a
description/comment of the fragment through
25　conventional data entry means. The fragment will then
be appended next to the selected node in the repository
tree under the entered name.

　　　The author can load a fragment from the fragment
repository into the current document within the editor.
30　To do so, the author places the cursor in the editor
where the fragment is to be inserted, then selects the
corresponding fragment the fragment repository tree and
triggers insertion. The fragment will then be inserted
505 at the correct place.

Further, for a fragment an XML file containing
metadata of the fragment can be generated. Source code
example 3 shows an example of such a fragment XML file.

5   Source code example 3:

```
<!DOCTYPE fragment SYSTEM "fragment.dtd">
<?xml version="1.0" encoding="UTF-8" ?>
<fragment:fragment>
     <fragment:name>Device-Class1</fragment:name>
     <fragment:parent>SearchResults</fragment:parent>
     <fragment:parent-
file>SearchResults.xml</fragment:parent-file>
     <fragment:comment></fragment:comment>
     <fragment:content>
<section           eccdc:deviceClassOneOf="DeviceClass1"
riml:frameId="table-content-frame">
            <table>
                    <tr>
                    <td class="BgBlue">
                            Michael Sting
                    </td>
                    </tr>
                    <tr>
                    <td class="BgBlue">
                            +49-7099-99-9999
                    </td>
                    </tr>
            </table>
     </section>
     </fragment:content>
</fragment:fragment>
```

Examples of metadata that can be included in
fragment XML files are: the fragment name, the fragment
parent (used by the tool for the structure of the

view), the fragment parent-file, fragment comments (entered by the author when saving a fragment), and the content that contains the verbatim of the saved fragment.

5   The IDE can further include a validation tool for evaluating the validity of the document 300. The XML standard allows one to specify XML schemas being in use in the current document 300 from the root element of the current document 300. For example, this root
10   element can be an HTML element. Locating XML schema files can be done by using the schemaLocation attribute from the XML schema namespace. For example, the RIML document 300 can be validated 506 against RIML schemas given in an HTML element. Thus, the author can quickly
15   identify errors and correct the errors on the basis of appropriate error messages 111-1 generated by the validation tool.

        If the document is valid, for example, a popup window can convey the message to the author. If the
20   document is invalid, a tasks view 111 of the IDE can be used to show the different errors. The task view 111 can be displayed simultaneously with the editor 104 and can be interfaced to the editor. When selecting an error 111-1 in the task view the editor highlights the
25   corresponding position 309 within the document 300 that has caused the error.

        Validation can be based on schema files given in the HTML element. The required RIML schemas can be found in a corresponding folder. The validation tool
30   can find the schemas using relative paths (e.g., relative to the RIML folder as a basis).

        FIG. 5 illustrates the IDE when further including a device dependent frames layouting view 124 to provide
35   to the author an overview of what the presentation

structure will look like for the various device
classes. Document pagination and transformation may be
disregarded by the frames layouting view. Similar to
frames in HTML, the author has a view of where various

5   portions of the document will be presented when the
document is displayed (e.g., how rows and columns of
the document will be distributed). Differences for the
various device classes can result from, for example,
using layout components that are tailored to a specific

10   device class and cannot be used for other device
classes.

For example, for device class DC1 (grey shaded
tab) the frames layouting view 124 shows a layout that
is composed of one column 124-0, which includes two

15   frames 124-1, 124-2. Each frame has one section
assigned to it. The highlighted section 312-1 in the
editor 104 corresponds to the selected element
(illustrated by "first section" in italics) in the
frames layouting view 124. This is achieved by

20   interfacing the editor 104 with the frames layouting
view 124 accordingly. For example, when the author
selects a section within first frame 124-1 the editor
highlights the corresponding text section 312-1. The
author can also select a section of the editor text

25   312-1 and the frames layouting view 124 will highlight
the corresponding frame 124-1.

FIG. 6 shows the IDE 999 further including the device
class dependent page view 122 to provide information to

30   the author about how the document 300 will be paginated
by the adaptation engine 105 for various device classes
DC1, DC2. Such a tool is described in the European
patent application 03024356.2.

For example, the author can start the adaptation

35   engine 105 from a corresponding menu of the IDE main

window. Adaptation engines, such as the consensus
adaptation engine, are known in the art. An adaptation
engine is used to generate device class specific
representations 301, 302 (cf. FIG. 8) of the document

5   300. In general, the document 300 includes a hierarchy
of layout components. This hierarchy can be adapted to
various device classes DC1, DC2 in different ways. This
may result in different representations 301, 302 of the
document 300 for various device classes DC1, DC2. For

10   example, a specific layout component may be suitable
for a first device class DC1 but not a second one DC2.
This specific layout component can be suppressed by the
adaptation engine 103 for the second device class DC2
and, therefore, does not become part of the document's

15   representation 302 for the second device class DC2. An
appropriate preview tool may allow the author to choose
a specific emulator for a preview. The output of the
adaptation engine 103 is generated for the chosen
emulator. The author can browse through generated sub-

20   pages in the preview of the document.

For generating the device-class dependent view 122
the adaptation engine may stop after the pagination
step. The output of the adaptation engine corresponds
to paginated pages (e.g., Page 1, Page 2) of the

25   document 300. As a result of this pre-pagination run of
the adaptation engine, the author can see where
different parts of the document 300 will be split,
dependent on the device class DC1, DC2.

The example of FIG. 6 shows a preview of how the

30   currently edited RIML document 300 will be paginated
for a first device class DC1 (grey shaded tab). For
example, the author can access the pagination previews
of the other device classes DC2 by selecting the
corresponding tab. The adaptation engine output

35   includes the content of each generated sub-page. For

17

example, the RIML document has at least two sub pages
(Page 1 and Page 2, further sub-pages can be accessed
by scrolling). The author can expand the nodes of a
sub-page so as to see their detailed content.

5   The device class dependent page view 122
integrates part of the adaptation engine 105 (up to the
pagination step) with the editor 104. For example, a
RIML document loaded in the editor 104 is provided to
the adaptation engine 105 and a set of generated sub-
10   pages Page 1, Page 2 is presented to the author. Thus,
the author is enabled through the device dependent page
view 122 to elaborate during the development of the
document 300 how specific changes to the document 300
will affect the appearance of the corresponding user
15   interface on various device classes. The author can
correct the document immediately when undesired
pagination results make the user interface unusable on
a specific device. If this would be done once the
document development is finished, a correction would
20   most probably imply major modifications to readjust
the document for compliance with all device classes.

FIG. 7 shows the editor 104 editing the document 300
when including Java code besides XML based code. To
25   handle such a document, the IDE 999 can further include
a Java filtering tool 108 to hide Java code when using
XML views, and to switch back to Java code when editing
Java. For example, inserting Java code into the RIML
document poses problems for XML view tool based editor
30   because Java code does not respect the XML rules and,
therefore, breaks the XML model that needs to be
respected for the XML tools to work normally.
       The author can activate a Java code view by using
the Java filtering tool 108 that allows editing Java
35   code. For example, the Java filtering tool can have a

corresponding graphical representation in the IDE main
window (e.g., a button or menu entry). For example,
Java code can be recognised by start characters "<%"
and end characters "%>". The Java filtering tool can
5   also handle JSP tag prefixes.

The author can switch back to the XML view to work
with XML tools. The Java code may be hidden in the
editor 104 when the Java filter is active. Java code
that is present prior to the HTML root element of the
10   document can be classified as a comment, as an XML
document doesn't allow any content prior to the root
element.

The Java filtering tool 108 has no impact on the
content of the document 300 when saved. The saved
15   document 300 includes the correct Java code.

The IDE 999 can further include a device class
dependent complexity indicator 121 as shown in FIG. 8.
The complexity indicator 121 has a complexity
20   evaluation library 121-1 for evaluating the complexity
of layout components used in the document 300 or its
device specific representations 301, 302 and further
has a complexity display 121-2 for visualizing the
result of the complexity evaluation. High complexity of
25   layout components usually has a negative impact on the
usability of the user interface that includes the
layout components.

The adaptation engine 105 receives 410 the
document 300 created 405 with the editor 104 as input
30   and generates 420 device specific representations of
the document considering specific constraints of a
device class (e.g., limited display area, memory
constraints). In the example, a first representation
301 is generated for device class DC1 and a second
35   representation 302 is generated for device class DC2.

Each representation can have a layout component hierarchy 321, 322 that is different from the one 320 of the original document 300. In the example, the adaptation engine removed layout component 4 when
5   generating the first representation 301 and layout component 3, when generating the second representation 302.

The complexity indicator 121 receives 430 information about layout components 1 to 9 and how
10 these layout components are built into the layout component hierarchies 321, 322 of the document representations 301, 302. A layout component can include multiple basic layout elements (e.g., input fields) and group these layout elements in such a way
15 that a specific function of the document (e.g., performing a search) is bundled in the layout component. Sometimes layout components are also referred to as controls.

The complexity indicator 121 determines the layout
20 components and the layout component hierarchy 321, 322 of the respective representation 301, 302.

Further, the complexity indicator 121 calculates a complexity value for each layout component in its respective representation 301, 302. This can be
25 achieved by using a complexity evaluation library 121-1 of the complexity indicator 121. It is sufficient that the complexity indicator can access the library 121-1, which may also be stored elsewhere within the IDE 999. The library 121-1 includes a set of complexity
30 evaluation functions EF5-DC1, EF5-DC2, EF6-DC1, EF6-DC2, etc. Preferably, such an evaluation function exists for each layout component with respect to the various device classes DC1, DC2. This can also be achieved by associating the evaluation functions with
35 specific layout component types, where each layout

component is an instance of the respective layout component type. The association of the evaluation functions with the respective layout components is illustrated by a solid line between a layout component

5   and its respective evaluation functions.

The complexity indicator 121 applies the evaluation functions for the various device classes to the layout components of the respective representations 301, 302. Each applied evaluation function returns a

10   complexity value for the respective layout component. For example, return values may range from 1 to 10, where 1 indicates a low complexity of the component and 10 indicates a high complexity of the component. Any other appropriate measure can be used instead.

15   Evaluation criteria used by the evaluation functions can, for example, refer to the number of items that can be displayed simultaneously in the display area of a specific device class or to the number of broken links of the layout component, dependent of the component

20   layout type.

Then, the complexity indicator aggregates the returned complexity values for the various representations 301, 302 according to the respective layout component hierarchies 321, 322. Aggregate

25   complexity values can be determined for the various levels in the layout component hierarchy 321, 322.

For example, layout component 2 represents a menu that includes two sub-menus (layout components 5 and 6). When applying the evaluation functions EF5-DC1 and

30   EF6-DC1 to the sub-menus 5, 6 for the first device class DC1 (first representation 301), the aggregation algorithm may propagate the maximum complexity value of both sub-menus to the menu 2, assuming that the complexity value of the menu 2 cannot be less than the

35   highest complexity value of is sub-menus. The same

applies to the second device class DC2 when applying the evaluation functions EF5-DC2 and EF6-DC2. However, even when both sub-menus 5, 6 have a low complexity value, the overall complexity of the menu 2 can still

5   be high. Therefore, in addition to propagating complexity values of child nodes in the layout component hierarchy to the parent node, an evaluation function can be applied directly to the parent node. For example, the sub-menus can have complexity values

10   of 3 and 5. However, the usage of both sub-menus in the menu 2 can lead to a complexity value 7 for the menu 2 (parent node) itself. Thus, the propagated complexity value of the sub-menus max(3;5)=5 would be overruled by the complexity indicator with the higher complexity

15   value 7 that is directly calculated for the parent node (menu 2).

The complexity indicator can then visualize 440 the various complexity values for the author in a complexity display 121-2. For example, the aggregate

20   complexity value for the respective component hierarchy 321, 322 can be displayed for each device class DC1, DC2.

In an alternative implementation, the complexity indicator processes complexity evaluation hierarchies

25   instead of layout component hierarchies. For this purpose, the IDE 999 can include a transformer that can transform the layout component hierarchy of each representation into a markup language independent complexity evaluation hierarchy. The complexity

30   evaluation hierarchy includes the same information as the respective layout component hierarchy but is described in a generic language to which the evaluation functions can be applied. Using a language independent complexity evaluation hierarchy enables the complexity

35   indicator to use a single set of evaluation functions

being associated with components of the complexity evaluation hierarchy in the complexity library 121-1. This association becomes independent from the markup language being used for the original document 300 or

5   its device specific representations 301, 302. The complexity hierarchy layer is an abstraction layer between the representations 301, 302 and the complexity indicator that helps to avoid that an evaluation function for a layout component needs to be redundantly

10   provided for various markup languages, such as RIML, XHTML, HTML, etc.

FIG. 9 shows an alternative implementation of the complexity display 121-2 when integrated into the IDE

15   main window.

    The complexity values for each device class DC1, DC2 are visualized as graphical bars. In the example, complexity values increase from the left value 1 to the right value 10. Threshold values T1, T2 are used to

20   change the appearance of the bar dependent on the visualized threshold value. For example, complexity values below T1 have a first grid structure or a first colour. Complexity values between T1 and T2 have a second grid structure or a second colour and complexity

25   values above T2 have a third grid structure or a third colour. Other presentations, such as traffic lights changing the colour when exceeding a threshold value, are also possible.

    The complexity display 121-2 further can have a

30   drill down section, where complexity values can be shown on different hierarchy levels down to the complexity of an isolated layout component for a selected device class. In the example, the drill down is made for the second device class DC2. Apparently,

35   the high complexity value originates from the layout

component 2, whereas the complexity value of layout components 4 and 7 is relatively low. A further drill down can be made for each of the layout components to determine the origin of high complexity values.

5   The tree-based outline editor 109 can be interfaced to the complexity indicator 121 so that, when it is displayed simultaneously, a layout component that is selected in the complexity display 121-2 is highlighted in the component hierarchy of the tree-

10   based outline editor 109. In this example, the tree-based outline editor 109 displays the layout component hierarchy of the respective representation 302 that corresponds to the device class DC2 that is currently drilled down in the complexity indicator.

15

Embodiments of the invention can be implemented in digital electronic circuitry, or in computer hardware, firmware, software, or in combinations of them. The invention can be implemented as a computer program

20   product, i.e., a computer program tangibly embodied in an information carrier, e.g., in a machine-readable storage device or in a propagated signal, for execution by, or to control the operation of, data processing apparatus, e.g., a programmable processor, a computer,

25   or multiple computers. An computer program for device dependent authoring of user interface documents can be written in any form of programming language, including compiled or interpreted languages, and it can be deployed in any form, including as a stand-alone

30   program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A computer program can be deployed to be executed on one computer or on multiple computers at one site or distributed across multiple sites and interconnected by

35   a communication network.

Method steps of the invention can be performed by one or more programmable processors executing a computer program to perform functions of the invention by operating on input data and generating output.

5 Method steps can also be performed by, and apparatus of the invention can be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application-specific integrated circuit).

10 Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and

15 data from a read-only memory or a random access memory or both. The essential elements of a computer are at least one processor for executing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be

20 operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto-optical disks, or optical disks. Information carriers suitable for embodying computer program instructions and data

25 include all forms of non-volatile memory, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto-optical disks; and CD-ROM and DVD-ROM disks.

30 The processor and the memory can be supplemented by, or incorporated in special purpose logic circuitry.

To provide for interaction with a user, the invention can be implemented on a computer having a display device, e.g., a cathode ray tube (CRT) or

35 liquid crystal display (LCD) monitor, for displaying information to the user and a keyboard and a pointing device, e.g., a mouse or a trackball, by which the user

can provide input to the computer. Other kinds of devices can be used to provide for interaction with a user as well; for example, feedback provided to the user can be any form of sensory feedback, e.g., visual feedback, auditory feedback, or tactile feedback; and input from the user can be received in any form, including acoustic, speech, or tactile input.

The invention can be implemented in a computing system that includes a back-end component, e.g., as a data server, or that includes a middleware component, e.g., an application server, or that includes a front-end component, e.g., a client computer having a graphical user interface or a Web browser through which a user can interact with an implementation of the invention, or any combination of such back-end, middleware, or front-end components. The components of the system can be interconnected by any form or medium of digital data communication, e.g., a communication network. Examples of communication networks include a local area network (LAN) and a wide area network (WAN), e.g., the Internet.

The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other.